Programación Python para Machine Learning

Lección 14: Deep Learning II.

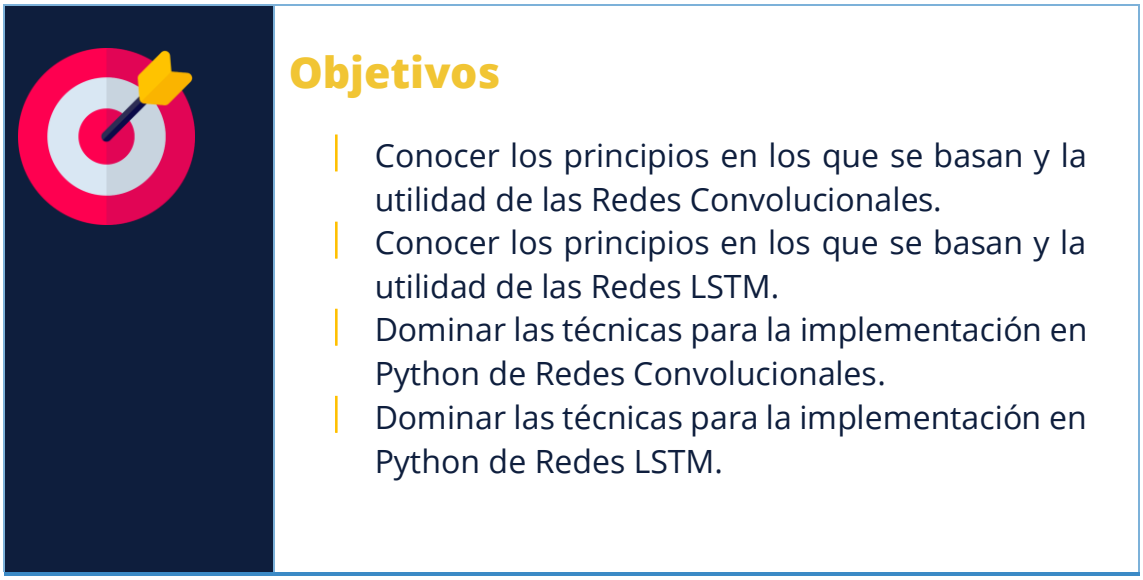
eip ESCUELA INTERNACIONAL POSGRADOS

ÍNDICE

Lección 14: Deep Learning II.

1. Introducción

El Deep Learning representa en la actualidad una de las áreas más prometedoras dentro del Machine Learning y la Inteligencia Artificial. La aparición y perfeccionamiento de este tipo de modelos pone a disposición de los desarrolladores un abanico de nuevas posibilidades de aplicación para la resolución de problemas supervisados y no supervisados. A lo largo de esta lección se estudiarán los principios básicos de dos modelos de Deep Learning, las Redes Convolucionales y las redes LSTM, así como se presenta el modo de implementar estos modelos utilizando las bibliotecas de Python TensorFlow y Keras para resolver problemas de predicción.



Objetivos

- Conocer los principios en los que se basan y la utilidad de las Redes Convolucionales.
- Conocer los principios en los que se basan y la utilidad de las Redes LSTM.
- Dominar las técnicas para la implementación en Python de Redes Convolucionales.
- Dominar las técnicas para la implementación en Python de Redes LSTM.

2. Redes Convolucionales

Una Red Neuronal Convolucional (CNN) es un modelo de arquitectura de red de Deep Learning que aprende directamente de los datos, sin necesidad de extraer características manualmente. Estas redes son particularmente útiles para encontrar patrones en imágenes, para reconocer objetos, caras y/o escenas.

Las Redes Convolucionales permiten detectar patrones en las instancias de entrada, normalmente imágenes, inspiradas en el modo de funcionamiento del sistema de visión humano. En el procedimiento de este tipo de redes, se entrenan diferentes capas capaces de extraer las características necesarias para posteriormente realizar la predicción.

Cada capa se especializa en detectar un tipo de patrón en la instancia, desde estructuras básicas, líneas, bordes o tonalidades de colores, pasando por capas intermedias capaces de identificar formas, hasta las capas más profundas que pueden identificar objetos o rostros. Las capas, conforme se va profundizando, se van especializando hasta que se logra detectar elementos más complicados dentro de la imagen. El principio de funcionamiento de una Red Convolucional se basa en la estructura matricial de las imágenes. Progresivamente, a través del entrenamiento de múltiples capas convolucionales, filtros que realizan distintas operaciones sobre la matriz, se extraen progresivamente características cada vez más complejas de la imagen para lograr su reconocimiento.

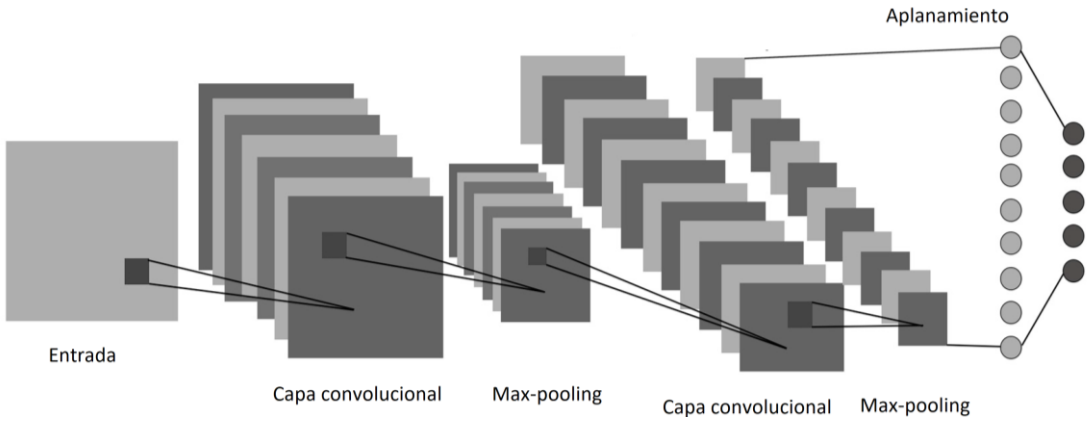


Ilustración 1: Estructura Red Convolucional

Como puede verse en la Ilustración 1, la instancia de entrada es procesada de manera progresiva por cada bloque de filtros de la red. En la primera capa, cada filtro tratará de detectar un elemento discriminante básico de la imagen. A la salida de ese bloque se le aplica un proceso de reducción, max-pooling, que realiza una selección de los datos mediante la extracción de la información más relevante obtenida en el paso previo. El procedimiento filtro-reducción se puede repetir un número determinado de veces, resultando cada vez un tamaño de imagen más pequeño, e incrementando la cantidad de filtros aplicados. Conforme las capas son más profundas en la red, se extraen más características a cada cual más sofisticada, siendo en última instancia combinadas para determinar la clase en la que se pretende clasificar. Aunque no siempre es así, la última capa es una red neuronal clásica con un número reducido de capas ocultas, una normalmente, que, dadas las características extraídas por las capas previas, se aplanan en un vector de entrada y se puede realizar la clasificación final de la imagen.

El éxito de las Redes Convolucionales radica precisamente en los filtros que se utilizan en cada capa y que permiten extraer diferentes características de la imagen. Dicho filtro consiste en una matriz, con una serie de coeficientes numéricos, que realiza un barrido a través de la imagen usando una operación matemática llamada convolución. Dependiendo de los coeficientes que tenga el filtro, tras esta convolución será posible detectar algunas características de la imagen de entrada.

Lo interesante de las Redes Convolucionales es que no es necesario diseñar manualmente cada uno de los filtros a utilizar. En su lugar, durante la fase de entrenamiento, la misma red aprende de forma automática, y con la ayuda de una gran cantidad de imágenes de entrenamiento, a ajustar iterativamente estos coeficientes para progresivamente ir detectando características cada vez más complejas en el proceso de entrada.

3. Implementación en Python de un modelo de Red Convolucional

Se tratará de mostrar la implementación con Keras de un modelo de Red Convolucional que sirva como clasificador capaz de determinar a qué clase corresponde una serie de imágenes. El conjunto de datos que se utilizará se denomina CIFAR-10 (véase Ilustración 2), y contiene un total de 60000 imágenes en color de 10 objetos distintos, 50000 de entrenamiento y 10000 de test, cada una de ellas con un tamaño de 32x32.

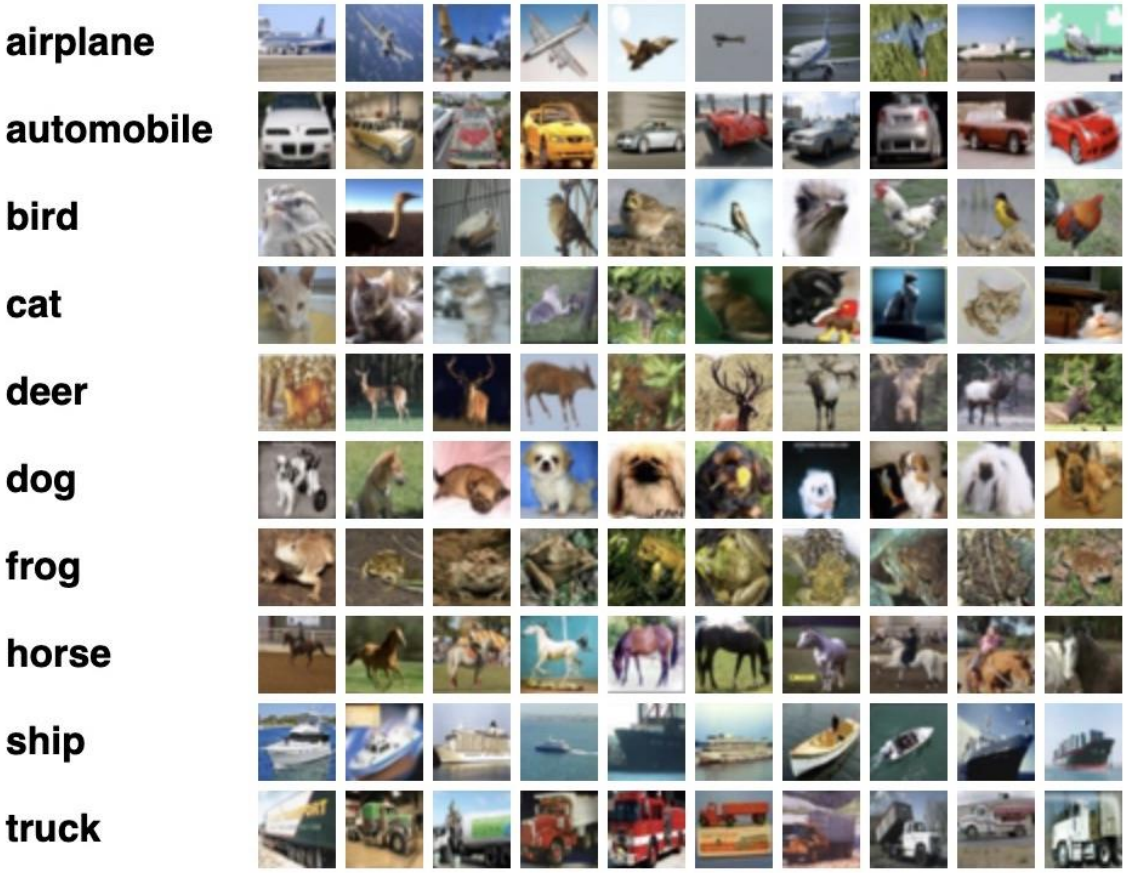


Ilustración 2: Categorías del conjunto CIFAR-10

La estructura de la red será la siguiente:

- Cada una de las instancias de entrada a esta red es una imagen de 32x32, que contendrá un objeto.

- La red está compuesta por una serie de capas convolucionales y de max-pooling. Progresivamente, se irán extrayendo las características más relevantes de cada imagen. Conforme se va profundizando, el tamaño de las imágenes resultantes va disminuyendo, pero se extraen más características de cada imagen. La salida de las capas convolucionales es un volumen de 6x6x16, que contiene las características más relevantes de las imágenes de entrenamiento.

- En el siguiente paso, el volumen de 6x6x16 de la etapa anterior se aplana en un vector de 2304x1, el cual sirve de entrada a una red neuronal completamente conectada que tiene una capa oculta de 64 neuronas.

- Finalmente, se determina la categoría de la imagen mediante el uso de la capa de salida, que es solo una función de activación softmax con 10 salidas correspondientes a cada una de las posibles categorías.

```
from tensorflow import keras
import random, time, pandas, numpy
from keras.datasets import cifar10
import matplotlib.pyplot as plt
from tensorflow.keras.layers import Flatten, Dense, Conv2D,
MaxPooling2D
from keras.utils import np_utils
```

```
seed=random.seed(time.time())

(X_train, y_train), (X_test, y_test) = cifar10.load_data()

class_num=numpy.unique(y_train).size

y_train=np_utils.to_categorical(y_train,class_num)
y_test=np_utils.to_categorical(y_test,class_num)

X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train = X_train / 255.0
X_test = X_test / 255.0


model = keras.Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu',
input_shape=(32, 32, 3)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))

model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(10, activation='softmax'))



model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
print(model.summary())

history = model.fit(X_train, y_train, validation_split=0.2, epochs=10,
batch_size=32)
scores = model.evaluate(X_test, y_test)
print("Accuracy: " , (scores[1]))

pandas.DataFrame(history.history).plot()
plt.show()
```

4. REDES LSTM

Las Redes LSTM son un tipo de red neuronal de múltiples capas que puede almacenar cierta información de instancias anteriores, permitiendo aprender secuencias de datos y generar otras. Por tanto, pueden ser especialmente útiles si se trabaja con datos de series de temporales.

Los modelos de Redes neuronales recurrentes básicas disponen de dos entradas. Por un lado, el patrón actual y, por otro, un estado anterior. A partir de ello, se obtienen dos salidas, la predicción y el valor actualizado del estado. Esta estructura permite a este tipo de modelos disponer de memoria de corto plazo. Sin embargo, para que los patrones anteriores tengan un efecto relevante en la predicción actual, la secuencia a procesar debe ser relativamente corta.

Los modelos de Redes Long Short Term Memory (LSTM) resuelven este inconveniente. Una Red LSTM tiene la capacidad de memorizar un dato importante en la secuencia preservándolo por varios intervalos de tiempo. Es por ello que se puede considerar que disponen de una memoria tanto de corto como de largo plazo. Funcionan de manera que tienen la capacidad de añadir o eliminar la información que consideren relevante para el procesamiento de la secuencia.

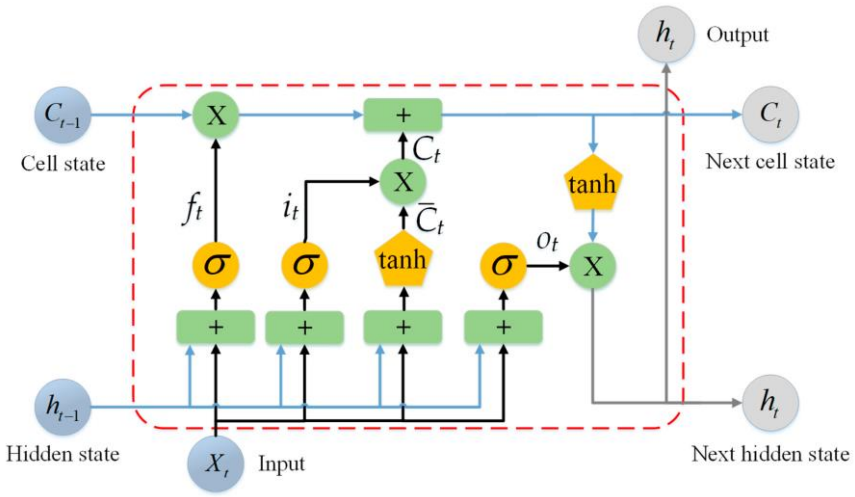


Ilustración 3: Estructura de una célula LSTM.

Las Redes LSTM están compuestas por células como la que se puede observar en la Ilustración 3. Cada célula LSTM tiene tres entradas y tres salidas. Las entradas se corresponden con el dato de entrada, el estado oculto en un tiempo t-1 y el estado de la célula en t-1. El estado es el elemento clave para el funcionamiento de las Redes LSTM. En el estado se pueden añadir o quitar datos que se desean disponer en la memoria de la red. Para añadir o quitar datos de dicha memoria, existen las llamadas puertas forget, que permite eliminar elementos de la memoria, la puerta update, que permite añadir nuevos elementos a la memoria y la puerta de salida, que crea el nuevo estado oculto. En realidad, estas puertas son redes neuronales que permiten o limitan el paso de información.

5. Implementación en Python de un modelo de Red LSTM

Se tratará de mostrar la implementación en Python de un modelo de Red LSTM que sirva como predictor capaz de determinar el valor de las acciones en bolsa de Repsol en un periodo de tiempo determinado. El conjunto de datos utilizado contiene registros diarios de la valoración de la acción de Repsol entre los años 2003 y 2021 tal y como se puede ver en la Ilustración 4. La estructura del modelo de red se implementa usando la capa LSTM de Keras.

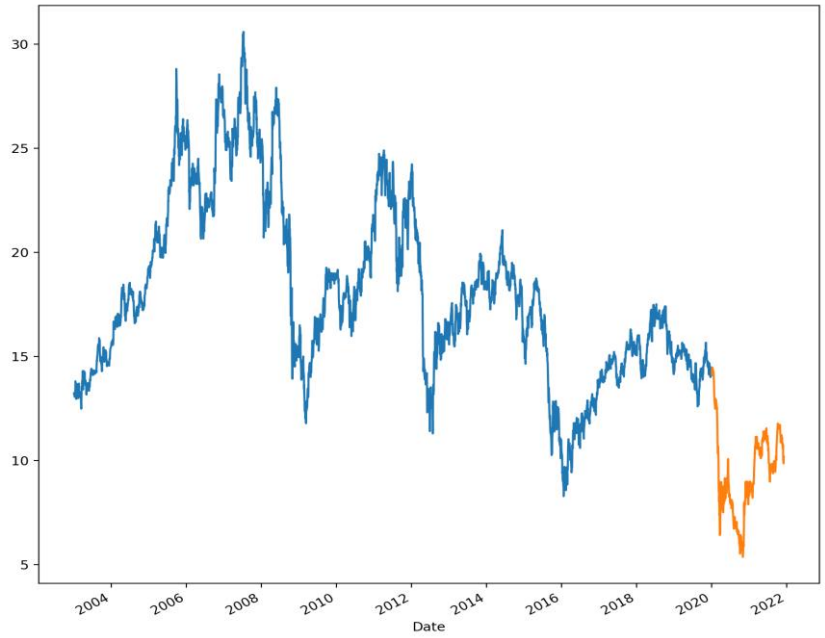


Ilustración 4: Evolución del precio de la acción de Repsol (2003-2021)

```
import numpy as np
np.random.seed(4)
import matplotlib.pyplot as plt
import pandas as pd

from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, LSTM
```

```
dataset = pd.read_csv('../datasets/REP.MC.csv', index_col='Date',
parse_dates=['Date'])
dataset.head()

dataset.dropna(inplace=True)

train = pd.DataFrame(dataset.High[:'2019'])
test =  pd.DataFrame(dataset.High['2020':])

dataset.High[:'2019'].plot()
dataset.High['2020':].plot()
plt.show()

minmaxSc = MinMaxScaler()
sc_train = minmaxSc.fit_transform(train)

windows = 100
X_train = []
Y_train = []
for i in range(windows, sc_train.size):
    X_train.append(sc_train[i-windows:i,0])
    Y_train.append(sc_train[i,0])

X_train, Y_train = np.array(X_train), np.array(Y_train)
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))

modelo = Sequential()
modelo.add(LSTM(units=60, input_shape=(X_train.shape[1],1)))
modelo.add(Dense(1))
modelo.compile(optimizer='adam', loss='mse')
modelo.fit(X_train,Y_train,epochs=10,batch_size=64)

sc_test = minmaxSc.transform(test)
X_test = []
for i in range(windows,sc_test.size):
    X_test.append(sc_test[i-windows:i,0])
X_test = np.array(X_test)
X_test = np.reshape(X_test, (X_test.shape[0],X_test.shape[1],1))

pred = modelo.predict(X_test)
pred = minmaxSc.inverse_transform(pred)
```

6. PUNTOS CLAVE

En esta lección se ha aprendido:

- Analizar los principios en los que se basan y la utilidad de las Redes Convolucionales.
- Estudiar los principios en los que se basan y la utilidad de las Redes LSTM.
- Utilizar las técnicas para la implementación en Python de Redes Convolucionales.
- Manejar las técnicas para la implementación en Python de Redes LSTM.